Safe Robot Execution in Model-Based Reinforcement Learning

David Martínez, Guillem Alenyà and Carme Torras

***Abstract*— Task learning in robotics requires repeatedly executing the same actions in different states to learn the model of the task. However, in real-world domains, there are usually sequences of actions that, if executed, may produce unrecoverable errors (e.g. breaking an object). Robots should avoid repeating such errors when learning, and thus explore the state space in a more intelligent way. This requires identifying dangerous action effects to avoid including such actions in the generated plans, while at the same time enforcing that the learned models are complete enough for the planner not to fall into dead-ends.**

We thus propose a new learning method that allows a robot to reason about dead-ends and their causes. Some such causes may be dangerous action effects (i.e., leading to unrecoverable errors if the action were executed in the given state) so that the method allows the robot to skip the exploration of risky actions and guarantees the safety of planned actions. If a plan might lead to a dead-end (e.g., one that includes a dangerous action effect), the robot tries to find an alternative safe plan and, if not found, it actively asks a teacher whether the risky action should be executed.

This method permits learning safe policies as well as minimizing unrecoverable errors during the learning process. Experimental validation of the approach is provided in two different scenarios: a robotic task and a simulated problem from the international planning competition. Our approach greatly increases success ratios in problems where previous approaches had high probabilities of failing.

I. Introduction

Robotic applications often include risky actions that, if applied under certain circumstances, may yield unrecoverable errors. When planning, these unrecoverable errors may lead to "dead-ends", as they are states from where the planner cannot provide a solution anymore [1]. The robot decision-maker should learn to identify these risky actions and be specially careful before executing them in order to avoid unrecoverable errors. This is a very challenging problem when learning tasks, since models not yet completely learned may lack important constraints and action effects needed to generate safer plans. In this paper, we propose a new method that extends active learning approaches to identify risky action and avoid repeatedly making the same errors.

An example of such domain would be the task of clearing tableware from a table, where the robot has to stack the tableware and move it to the kitchen. As moving between different rooms takes a long time, it is recommended to stack all objects together before, and take piles with as many objects as possible when moving. The problem is that, when tableware is not properly stacked, objects may fall and break. Thus, the robot needs a decision maker that plans which objects should be stacked, and the correct order so the task can be completed safely. Figure 1 shows the task of stacking tableware performed by a WAM manipulator.

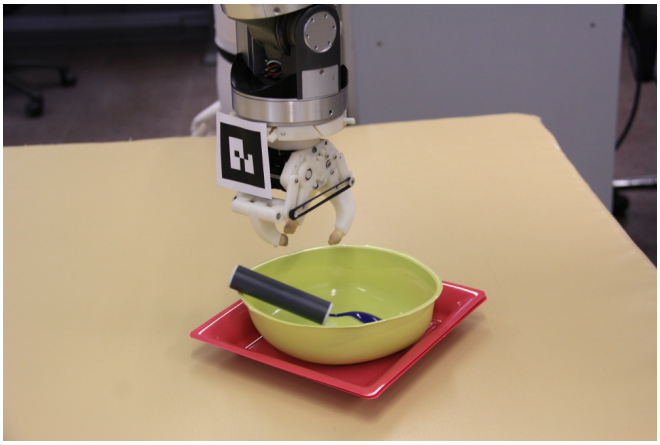

Fig. 1. The WAM manipulator is stacking the tableware to clear the table.

The decision maker uses model-based reinforcement learning (RL), which is a common paradigm for learning tasks in which actions take long to execute and the learning algorithm has long periods of time to process little input data [2]. However, the learning process requires exploring (i.e. trying different action-state pairs) to learn the transition function [3], [4]. This can be a problem when tackling stochastic domains, since the same dead-end may be reached repeatedly before learning all the conditions that lead to it. The decision maker has to recognize risky actions to ensure that they are not explored when there is risk of falling into a dead-end. In the opposite case, when using fast learning algorithms that have good generalization capabilities to reduce the exploration [5], the robot may overlook crucial actions required to get an optimal behavior, and learn simple policies that often fall into dead-ends.

Safety mechanisms have been included in different RL approaches to avoid these risky actions. These approaches either require a safety function to avoid dangerous states while exploring [6], [7], or use simulators to confirm the safety of states [8]. In our case, the decision maker does not have any prior information about the risky states that may be found, but in contrast, every time a dead-end is reached, it will learn the causes and avoid them in the future.

A general method to analyze dead-end causes is using planning *excuses* [9], [10], which are state changes that

This work was supported by CSIC project MANIPlus 201350E102. D. Martínez is also supported by the Spanish Ministry of Education, Culture and Sport via a FPU doctoral grant (FPU12-04173).

Authors are with Institut de Robòtica i Informàtica Industrial, CSIC-UPC, Llorens i Artigas 4-6, 08028 Barcelona, Spain {dmartinez,galenya,ctorras}@iri.upc.edu

would have to be made to find a proper plan. In this work we propose to use these planning excuses to detect state predicates associated to dead-ends (which we call "dangerous predicates"), that will be used to detect risky actions. Note that excuses require a model, and thus our approach can only be applied to model-based RL.

Teacher interactions have been integrated with RL to speed up the learning process [11], [12], as well as to provide corrective demonstrations when the robot does not perform as desired [13], [14]. In this work we use the REX-D algorithm [15] which requests only a few teacher demonstration requests to speed up learning. To avoid dead-ends, the interaction with the teacher is extended to confirm risky actions. Once a risky action has been identified, if a plan includes it, the decision maker can issue a teacher interaction request to confirm its safety or request alternatives.

To summarize, the proposed learning approach yields safer models and avoids dead-ends. Unlike other approaches, our method can rely on models that are not completely learned yet. Moreover, the robot is the one that actively interacts with the teacher to learn how to overcome the dead-ends, contrarily to other approaches that require the teacher to continuously monitor the robot behavior. A preliminary version of this work was presented in [16]. The dead-end avoidance method has been improved significantly: the decision maker searches for alternative plans before requesting an interaction with the teacher, and the acceptable risk of dead-end is maintained for each type of unrecoverable error (while previously a global acceptable risk was used for every dead-end). Finally, in addition to simulated experiments, the performance in a real robot is also shown.

The paper is organized as follows. After this introduction, the background needed to understand our proposal is presented in Section II. Afterwards, the method to avoid dead-ends is detailed in Section III. Section IV shows the experiments conducted in a simulator and a real robot. Finally, conclusions and future work are presented in Section V.

II. The Decision Making and Learning Algorithm

The initial assumption is that the perception modules can provide full observability, and that robot actions are uncertain because they may fail or yield unexpected outcomes. Therefore, **Markov Decision Processes** (MDP) can be used, as they formulate fully-observable problems with uncertainty. A finite MDP is a five-tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \alpha \rangle$ where $\mathcal{S}$ is a set of possible discrete states, $\mathcal{A}$ is the set of actions that the robot can execute, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function, $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function and $\alpha \in [0, 1)$ is the discount factor. The goal is to find a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that chooses the best action for each state to maximize future rewards. To that end, we have to maximize the value function $V^{\pi}(s) = E[\sum_t \alpha^t R(s_t, a_t) \mid s_0 = s, \pi]$, which is the sum of expected rewards.

The decision maker uses a symbolic representation to represent the model and the states. A state is composed of a set of symbolic predicates that represent the scenario that the robot is interacting with. The model consists of a set of **Noisy Indeterministic Deictic (NID)** rules [17]. The transition model T is represented with a set of NID rules Γ which are defined as:

Action:
putPlateOn(X, Y)
Preconditions:
plate(X), on(X,Z), cup(Y), clear(Y)
Outcomes (Success probability: predicate changes):
0.6: on(X,Y), ¬on(X,Z), clear(Z), ¬clear(Y)
0.3: on(X,Y), ¬on(X,Z), clear(Z), ¬clear(Y), unstable(X)
0.1: noise

Fig. 2. NID rule example that models one behavior of *putPlateOn* action.

$$a_r(\chi) : \phi_r(\chi) \rightarrow \begin{cases} p_{\Omega_{r,1}} : \Omega_{r,1}(\chi) \\ \vdots \\ p_{\Omega_{r,n_r}} : \Omega_{r,n_r}(\chi) \\ p_{\Omega_{r,0}} : \Omega_{r,0} \end{cases}, \tag{1}$$

where a_r is the action that the rule represents, $\phi_r(\chi)$ are the preconditions for the rule to be applicable, $\Omega_{r,i}$ are the effects that define the set of predicates that are changed in the state with probability $p_{\Omega_{r,i}}$ when the rule is applied, $\Omega_{r,0}$ is the noisy effect that represents all other, unmodeled, rare and complex effects, and χ is the set of variables of the rule. An example of a NID rule is shown in fig. 2. A NID rule represents one action, while each action may be represented by several rules. Each state-action pair (s, a) is covered by just one rule r as all the rules that define one action have disjoint preconditions $\phi_{r_{a,i}} \wedge \phi_{r_{a,j}} = \emptyset \mid \forall i, j, i \neq j$. Finally, a context is defined as the state-action pairs covered by a rule. Note that, as actions are modeled by rules, the presented method in this paper finds the dangerous rules that represent risky contexts.

Reinforcement learning permits learning a transition model T which is unknown a priori. A RL algorithm balances exploration (try different actions to increase the decision maker knowledge and obtain better policies in the long term) and exploitation (choose actions to maximize the reward according to the current policy) to obtain good results. In particular we are using model-based RL where a model is estimated from experiences, and this model is then used to plan the actions that the system executes.

In particular, we are using the **REX-D** [15] algorithm that integrates active teacher demonstration requests to speed up learning. REX-D balances exploration, exploitation and teacher demonstrations. To that end, REX-D explores the state-action pairs considered unknown before exploiting the model to try to obtain the maximum value. Moreover, if no solution is found in a known state, then REX-D requests a teacher demonstration as it considers that a completely new action or yet unknown effects of an action under different preconditions need to be demonstrated. Figure 3 shows a summary of how the demonstration requests are integrated in the RL algorithm.

REX-D uses a relational representation [5], generalizing over different objects of the same type as they exhibit the

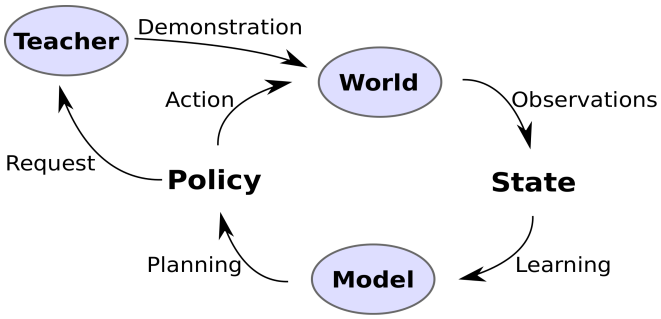

Fig. 3. The REX-D algorithm. Teacher demonstrations are requested by the robot when the planner cannot obtain a valid solution.

same behavior. A context-based density count function is used to handle the exploration-exploitation dilemma, which reduces the number of samples before considering states as known by grouping them in contexts:

$$k(s,a) = \sum_{r \in \Gamma} |E(r)| I(r = r_{s,a}), \tag{2}$$

where $|E(r)|$ is the number of experiences that cover the rule with any grounding, and $I()$ is a function that takes the value 1 if the argument evaluates to true and 0 otherwise. State-action pairs which have been explored less times than a fixed threshold $k(s,a) < \zeta$ are considered unknown, and the REX-D algorithm proceeds to explore them.

Finally, to tackle stochastic environments Pasula et al.'s learning algorithm [17] is used to obtain the rules that represent the actions from all previous experiences, and the Gourmand planner [18] is used to plan with the model and the observed state.

III. Proposed Approach to Avoid Dead-ends

In this section we propose a new method that allows the decision maker to avoid exploring dangerous parts of the state space as well as to safely refine such parts of the model, so that the planner can avoid falling in dead-ends. The following shortcomings are overcome with dead-end avoidance:

- Exploration of dead-ends: until the states that lead to a unrecoverable error get explored, a RL algorithm would fall into that dead-end repeatedly.
- Suboptimal policies: Introducing relational generalizations and teacher demonstrations has the advantage of reducing the exploration [15], but it also has drawbacks. Using a relational count function implies that not all states are explored before considering them as known, since all states within a context are assumed to behave likewise. Therefore, state-action pairs that could be needed to attain the best policy might not be visited, and thus their contexts would not be learned. This lack of exploration may lead to models that yield suboptimal policies that fall more frequently into dead-ends.

First, we explain how the dangerous predicates that model unrecoverable errors are detected, and how these dangerous predicates are used to find dangerous rules. Then we describe how the decision maker analyzes plans to check whether they have a high-risk of reaching a dead-end. Finally the procedures to avoid dead-ends are presented.

A. Detecting Dangerous Predicates

The first requirement to avoid unrecoverable errors is the ability to detect their causes. In the proposed model, a dead-end has to be caused by one or more predicates that, when present, prevent the planner from finding a solution. We propose to find these predicates using the so called *excuses* [9]. Excuses are defined as changes to the state that turn the task into a solvable one. In a more formal manner, given an unsolvable planning task that involves a set of objects C_π and an initial state s_0, an excuse is a pair $\varphi = \langle C_\varphi, s_\varphi \rangle$ that makes the task solvable, where C_φ is a new set of objects and s_φ a new initial state. However, as we are not considering domains in which objects can be added or removed, we will only focus on the state changes s_φ.

Excuses can be classified as acceptable, good, or perfect, as follows.

- *Acceptable excuses* change the minimum number of predicates in the initial state. An excuse φ is acceptable iff $\forall \varphi', C_\varphi \subseteq C_{\varphi'}$ and $s_0 \triangle s_\varphi \subseteq s_0 \triangle s_{\varphi'}$ (where $\triangle$ denotes the symmetric set difference).
- *Good excuses* are acceptable excuses with changes that cannot be explained by another acceptable excuse.
- A *perfect excuse* is a good excuse that minimizes a cost function.

Applying goal regression over all acceptable excuses is highly suboptimal, so a set of good excuse candidates are generated. Candidates for good excuses can be obtained under certain assumptions using a causal graph and a domain transition graph [9]. Also, to restrict the number of candidates, we only consider those that are relevant for achieving the goal. Finally, the planner is used to test which of the excuse candidates are the best. For each candidate, a new rule that encodes the excuse changes is added, and the planner selects the best excuse with the minimal cost.

From a planning point of view, the dangerous predicates are those that when present in the state, the planner can no longer find a sequence of actions that reaches the goal. Therefore dangerous predicates represent unrecoverable errors. If an excuse is obtained in a state with an unrecoverable error, the excuse changes will show the dangerous predicates that are making the task unsolvable. Thus the list of dangerous predicates P_φ can be extracted as $P_\varphi = s \triangle s_\varphi \mid \langle C_\varphi, s_\varphi \rangle = \varphi$, where s is the current state and $\triangle$ denotes the symmetric set difference.

1) Library of Dangerous Predicates: The decision maker maintains a library of dangerous predicates L_d. This library contains a list of pairs $\langle d_p, d_r \rangle$ where d_p is the predicate that has caused an unrecoverable failure and d_r the acceptable risk for that predicate. The acceptable risks are always initialized to 0. This library is updated or used in the following cases:

- Whenever a dead-end is reached: an excuse φ is obtained, and for each predicate p changed by the excuse

Algorithm 1 Avoid dangerous rules

Input: Planned dangerous rule r, current state s, library of dangerous predicates L_d

1: $L'_d = L_d$ ▷ *Store expected dead-end risk*
2: $d'_r = 0 \, \forall \langle d'_p, d'_r \rangle \in L'_d$ ▷ *Initialize expected risks to 0*
3: $danger = false$
4: **for** each effect $\langle \Omega_r, p_{\Omega_r} \rangle$ in rule r **do**
5: $s' = s$
6: Add predicates in Ω_r to s'
7: Plan(s')
8: **if** dead-end **then**
9: **for** each dangerous predicate $d_p \in \Omega_r$ **do**
10: Get $d'_r \mid \langle d'_p, d'_r \rangle \in L'_d$, $d_p = d'_p$
11: Update $d'_r = d'_r + p_{\Omega_r}$
12: **if** $d'_r > d_r \mid \langle d_p, d_r \rangle \in L_d$ **then**
13: $danger = true$
14: **end if**
15: **end for**
16: **end if**
17: **end for**
18: **if** $danger$ **then**
19: Plan without rule r
20: **if** Obtained new plan **then**
21: Analyze new plan
22: **else**
23: Request teacher confirmation
24: **if** confirmed as safe **then**
25: **for** each $\langle p'_d, d'_r \rangle$ in L'_d **do**
26: Get $d_r \mid \langle d_p, d_r \rangle \in L_d$, $d'_p = d_p$
27: $d_r = max(d_r, d'_r)$
28: **end for**
29: **else**
30: Get demonstration from the teacher
31: **end if**
32: **end if**
33: **end if**

$p \in P_\varphi$, we add the pair $\langle p, 0 \rangle$ to the library if p had not been added before $\nexists p \mid \langle p, d_r \rangle \in L_d$.

- Whenever the teacher confirms that a dangerous rule has an acceptable risk: the risk associated to the predicates in the rule effects are updated (Sec. III-B.2).
- Whenever the model Γ is updated: any rule $r \in \Gamma$ whose effects include a predicate $p \in \Omega_r \mid \langle p, d_r \rangle \in L_d$ is marked as dangerous.

B. Avoiding Dead-ends

To avoid dead-ends, the decision maker has to skip exploration of potentially dangerous rules and request help from the teacher when the learned model leads to dead-ends.

1) Safe Exploration: To explore an action, rules have to be considered known (using Eq. 2) and safe. To check the safety, for each $\langle d_p, d_r \rangle \in L_d$ the decision maker checks that the risk for each dangerous predicate is low enough. The sum of the probabilities of getting the dangerous predicate $p(d_p \mid r) = \sum p_{\Omega_r} \, \forall \{ \langle \Omega_r, p_{\Omega_r} \rangle \in r \mid d_p \in \Omega_{r,i} \}$ has to be lower than the acceptable risk d_r for that predicate d_p.

2) Learning Safe Models: Whenever a new action is planned, the decision maker checks if the state-action pair corresponds to a dangerous rule. In this case a special procedure is executed to analyze the action safety.

Algorithm 1 summarizes the procedure to detect cases where there is danger of falling into a dead-end. First, in lines 4-11, the possible effects Ω_r of the planned dangerous rule r are simulated, and the planner is used to check whether any of them may lead to an dead-end. After this analysis, the decision maker knows the expected dead-end probability d'_r associated to each dangerous predicate p'_d. In lines 12-14, if the expected dead-end probability is greater than the acceptable risk for any predicate $d'_r > d_r \; \forall \{ d_p, d_r, d'_r \mid \langle d_p, d_r \rangle \in L_d, \langle d_p, d'_r \rangle \in L'_d \}$, either an alternative plan is found or teacher help is required.

After finding a risky rule, the decision maker plans again using a model that does not contain the dangerous rule r (lines 18-21). If a safe plan is obtained, it can be executed, and otherwise it is also analyzed using Algorithm 1. However, if a plan cannot be found, the robot interacts with the teacher to confirm the safety of the originally planned action (lines 22-28). The confirmation consists in notifying the teacher about the action that the robot intends to execute, and the teacher can either confirm that it is safe or demonstrate a different and safer action instead. To handle domains where dead-ends are not completely avoidable, if the teacher confirms a rule as safe, the risks associated to the dangerous predicates in that rule are updated $d_r = max(d_r, d'_r + \epsilon)$, considering that there was an estimated d'_r risk of dead-end for that predicate. However, if the rule is actually dangerous (lines 29-31), a safer action will be presumably demonstrated by the teacher. This new action will be learned and added to the model, so that the planner will have the option to choose it afterwards.

IV. Experimental Results

Two experiments were carried out to validate our proposal. The first one was a simulated problem from the Triangle Tireworld domain of the 2008 International Probabilistic Planning Competition (IPPC)The second was a real robotic task, which consisted in clearing the tableware on a table.

A. IPPC: Triangle Tireworld

In this domain, a car has to move to its destination, but it has a probability $p = 35\%$ of getting a flat tire while it moves. The car starts with no spare tires but it can pick them up in some locations. The actions available in this domain are: a "Move" action to go to an adjacent position, a "Change Tire" action to replace a flat tire with a spare tire, and a "Load Tire" action to load a spare tire into the car if there are any in the current location. The main difficulty in the Triangle Tireworld domain is the unrecoverable error when the agent gets a flat tire and no spare tires are available. Safe and long paths exist with spare tires, but the shortest paths do not have any spare tires. If the robot does not get

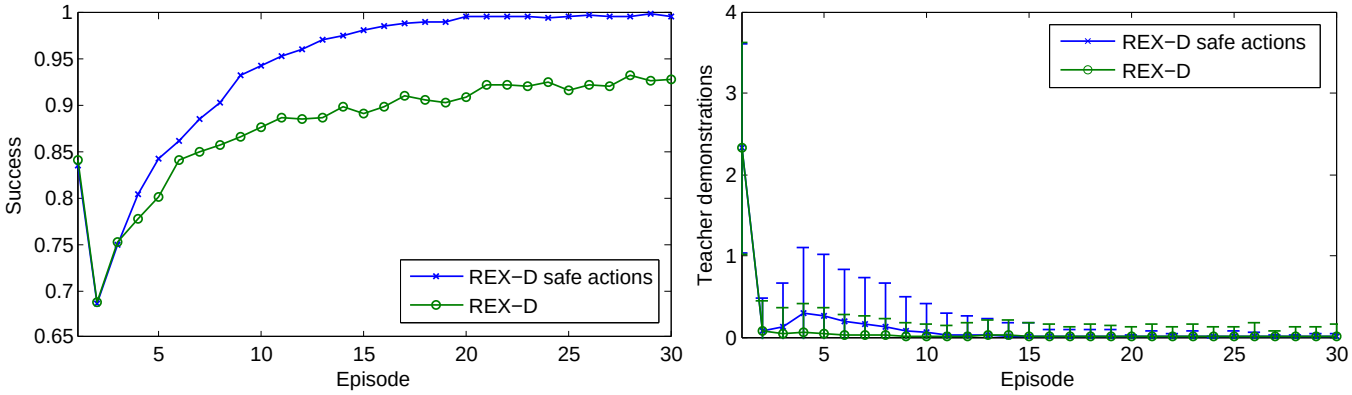

Fig. 4. Results in the Triangle Tireworld domain. The results shown are the means and standard deviations obtained from 2500 runs. The exploration threshold is $\zeta = 2$. **Left:** the success ratio of the algorithms. **Right:** the number of teacher demonstrations requested.

a flat tire while it explores locations where spare tires are available, it will not learn how to change a tire and therefore it will always choose the shortest path to the goal where no spare tires are available. It is a challenging domain where RL approaches with a low exploration threshold ζ can fall easily into recurrent dead-ends.

However, the dead-end avoidance permits learning correctly this problem without requiring a large exploration threshold and thus not increasing the number of exploration actions needed to learn the domain. After the robot realizes that moving may lead to flat tires, it will confirm the moving actions with the teacher, who will recommend the safer paths with spare tires instead of the shorter ones. Once the car gets a flat tire with a spare tire available, it will learn how to change tires and perform successfully afterwards. Figure 4 shows the advantages of recognizing and dealing with dangerous rules, where a few extra teacher demonstration requests allow the robot to have a much better success ratio, getting a 98% success ratio after 15 episodes, compared with 88% that would have been obtained without considering dangerous rules (Fig. 4 Left). The cost of this improvement is very low, as just one extra teacher demonstration is needed in average (Fig. 4 Right).

B. Tableware Clearing

This experiment consists in clearing the tableware on a table. The robot has to take plates, cups and spoons from a table to the kitchen. However, moving to the kitchen is a costly action, and therefore the robot has to stack the tableware before taking them, minimizing the time spent.

The actions *putPlateOn*, *putCupOn* and *putForkOn* place the corresponding object on another one. If a plate is placed on a cup or spoon, or a cup is placed on a spoon, it may fall and break. (For practical reasons, we used plastic tableware in the experiments so that they wouldn't actually break. Instead, they were considered to be broken after falling from a considerable height.) There are another 3 inverse actions to place each of the objects back on the table, and one action *moveStacks* to take the stacks to the kitchen.

The reward is based on the time spent, but there is also a high penalty for each broken object. Each action reduces the reward by 0.05, while the *moveStacks* action reduces it by 1 for every stack, and by 5 for every broken object. Finally, when the table is cleared, a reward of 5 is obtained.

The robotic setup consists of a WAM arm equiped with a gripper, and a RGB-D Kinect camera that is positioned on the ceiling. To generate symbolic state representations for the decision maker, the perception system recognizes the tableware on the table and their relative positions, and maintains a believe state that is needed to tackle the occlusions when an object is placed on top of another. The movement primitives to execute pick and place actions are also available in the robot. Overall, the implementation is similar to the one presented in [19], but the vision system and action trajectories have been adapted to the new task. Moreover, the interaction with the teacher is implemented using a PC located near the robot.

In this experiment two different models were learned using the REX-D algorithm [15]: one was learned with the dead-end avoidance method presented in this paper, and the other with the standard REX-D algorithm. Both were learned during 8 episodes in which the teacher was available to answer the robot inquiries. The initial setup consisted of 2 plates, 2 cups, and 1 spoon as shown in Fig. 5.

The two models were then used to complete the tableware clearing task 15 times, and the results are shown in Fig. 6. The model with dead-end avoidance reduced the number of executions with broken objects drastically, from 40% to only 13%. Note that robot actions and perceptions are not perfect, and reducing more the number of broken objects would be very complicated without improving these actions.

Moreover, the model with dead-end avoidance did not always stack everything into a single pile. The reason is that the risk of breaking an object (and thus falling into a dead-end) was not worth stacking the two piles together. Therefore the decision maker decided to finish with two piles in 20% of the runs.

V. Conclusions

We have presented a method that reduces the amount of unrecoverable errors when using model-based RL, and that

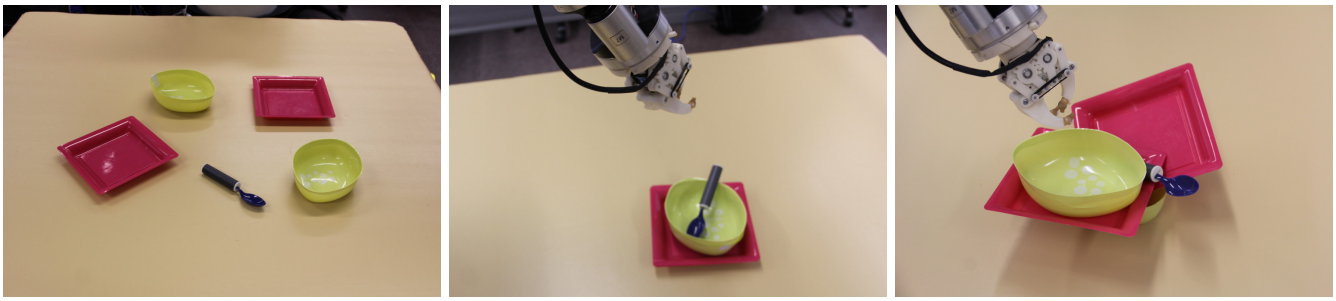

Fig. 5. Tableware clearing setup. **Left:** Example of the initial setup. **Middle:** Example of a perfectly stacked pile of tableware. **Right:** Example of a badly stacked pile, which if moved will result in objects falling from the pile.

REX-D	Success 1 pile (reward)	Success 1 or 2 piles (reward)	Reward all cases	Broken object
Dead-end avoidance	67% (3.78 ± 0.02)	87% (3.54 ± 0.43)	2.56 ± 2.79	13%
Standard REX-D	60% (3.58 ± 0.13)	60% (3.58 ± 0.13)	-1.61 ± 7.49	40%

Fig. 6. Results for clearing tableware. The results shown are the success ratios, and the means and standard deviations of the accumulated rewards obtained from 15 runs. The scenario was initialized with 2 plates, 2 cups, and 1 spoon. The exploration threshold is $\zeta = 3$. **Success 1 pile:** Experiments where the robot successfully (i.e. with no broken objects) piled everything into one pile (and the accumulated reward obtained in the successful executions). **Success 1 or 2 piles:** The same as before, but obtaining 1 or 2 piles. **All:** Accumulated reward counting all the executions. **Broken objects:** Experiments where the robot broke at least one object.

also ensures that safe models are learned even in cases with very sparse exploration. The robot is able to reason about dead-ends, analyze the causes, avoid them in the easier cases, and interact with a teacher to solve more complicated ones. In addition to learning how to avoid dead-ends, the robot is the one actively monitoring the scenario and interacting with the teacher when needed, releasing the teacher from monitoring the robot continuously. Experimental results have proven the advantages of the method, both in an IPPC domain and a robotics scenario. Our approach was shown to yield good success ratios with just a few extra teacher interactions.

As future work, we would like to extend the approach to work under partial observability, which is an ubiquitous problem in robotics. This would require to improve the analysis of dead-ends to consider candidates that may not have been observed, and integrate a POMDP planner that could tackle partial observations.

VI. Acknowledgements

We would like to thank Alberto Ezquerro for helping us to perform the robotic experiments with the WAM arm.

References

[1] I. Little and S. Thiebaux, "Probabilistic planning vs. replanning," in *Proc. of the ICAPS Workshop on IPC: Past, Present and Future*, 2007.

[2] J. Kober and J. Peters, "Reinforcement learning in robotics: A survey," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[3] R. I. Brafman and M. Tennenholtz, "R-max-a general polynomial time algorithm for near-optimal reinforcement learning," *Journal of Machine Learning Research*, vol. 3, pp. 213–231, 2003.

[4] A. L. Strehl, L. Li, and M. L. Littman, "Reinforcement learning in finite mdps: Pac analysis," *Journal of Machine Learning Research*, vol. 10, pp. 2413–2444, 2009.

[5] T. Lang, M. Toussaint, and K. Kersting, "Exploration in relational domains for model-based reinforcement learning," *Journal of Machine Learning Research*, vol. 13, pp. 3691–3734, 2012.

[6] A. K. Akametalu, S. Kaynama, J. F. Fisac, M. N. Zeilinger, J. H. Gillula, and C. J. Tomlin, "Reachability-based safe learning with gaussian processes," in *Proc. of the Conference on Decision and Control*, 2014, pp. 1424–1431.

[7] T. M. Moldovan and P. Abbeel, "Safe exploration in markov decision processes," in *Proc. of the International Conference on Machine Learning*, 2012, pp. 1711–1718.

[8] M. Pecka, K. Zimmermann, and T. Svoboda, "Safe exploration for reinforcement learning in real unstructured environments," in *Proc. of the Computer Vision Winter Workshop*, 2015.

[9] M. Göbelbecker, T. Keller, P. Eyerich, M. Brenner, and B. Nebel, "Coming up with good excuses: What to do when no plan can be found." in *Proc. of the International Conference on Automated Planning and Scheduling*, 2010, pp. 81–88.

[10] M. V. Menezes, L. N. de Barros, and S. do Lago Pereira, "Planning task validation," in *Proc. of the ICAPS Workshop on Scheduling and Planning Applications*, 2012, pp. 48–55.

[11] A. Agostini, C. Torras, and F. Wörgötter, "Integrating task planning and interactive learning for robots to work in human environments." in *Proc. of the International Joint Conference on Artificial Intelligence*, 2011, pp. 2386–2391.

[12] S. Chernova and M. Veloso, "Interactive policy learning through confidence-based autonomy," *Journal of Artificial Intelligence Research*, vol. 34, no. 1, pp. 1–25, 2009.

[13] Ç. Meriçli, M. Veloso, and H. L. Akın, "Multi-resolution corrective demonstration for efficient task execution and refinement," *International Journal of Social Robotics*, vol. 4, no. 4, pp. 423–435, 2012.

[14] T. J. Walsh, K. Subramanian, M. L. Littman, and C. Diuk, "Generalizing apprenticeship learning across hypothesis classes," in *Proc. of the International Conference on Machine Learning*, 2010, pp. 1119–1126.

[15] D. Martínez, G. Alenyà, and C. Torras, "Relational reinforcement learning with guided demonstrations," *Artificial Intelligence Journal, Special issue on AI and Robotics*, 2015.

[16] D. Martínez, G. Alenyà, and C. Torras, "Finding safe policies in model-based active learning," in *Proc. of the IROS workshop on Machine Learning in Planning and Control of Robot Motion*, 2014.

[17] H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling, "Learning symbolic models of stochastic domains," *Journal of Artificial Intelligence Research*, vol. 29, no. 1, pp. 309–352, 2007.

[18] A. Kolobov, Mausam, and D. S. Weld, "Lrtdp versus uct for online probabilistic planning," in *Proc. of the AAAI Conference on Artificial Intelligence*, 2012, pp. 1786–1792.

[19] D. Martínez, G. Alenyà, and C. Torras, "Planning robot manipulation to clean planar surfaces," *Engineering Applications of Artificial Intelligence*, vol. 39, pp. 23–32, 2015.